Java 语言讲义

欢迎来到Java语言讲义。这里我们重点讨论Java的入门学习和基本的语法，场景。

整个讲义基于`Java 7`

《Java 编程思想》，Think in Java

Java 语言概述

Java 发展简史

Java诞生于1995年，是Sun公司组织开发的一种编程语言，主要贡献者是James Gosling。

Java分为三个体系：

- Java SE (Java Platform Standard Edition，Java平台标准版）
- Java EE (Java Platform Enterprise Edition，Java平台企业版)
- Java ME (Java Platform Micro Edition，Java平台微型版)

`JDK`, Java Development Kit, Java语言开发工具包

2010年

- Oracle收购Sun公司以及产品，包括Java
- Steve Jobs声称，苹果公司将来不再支持Java

Java 语言特性

- 面向对象
    - Java语言提供类、接口和继承等原语，为了简单起见，只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为implements）。Java语言全面支持动态绑定，而C++语言只对虚函数使用动态绑定。总之，Java语言是一个纯的面向对象程序设计语言。
- 平台无关
    - Java程序（后缀为java的文件）在Java平台上被编译为体系结构中立的字节码格式（后缀为class的文件），然后可以在实现这个Java平台的任何系统中运行。这种途径适合于异构的网络环境和软件的分发。
    - 支持Mac，Windows，Linux，UNIX
- 多线程
    - 在Java语言中，线程是一种特殊的对象，它必须由Thread类或其子（孙）类来创建。通常有两种方法来创建线程：其一，使用型构为Thread(Runnable)的构造子将一个实现了Runnable接口的对象包装成一个线程，其二，从Thread类派生出子类并重写run方法，使用该子类创建的对象

即为线程。值得注意的是Thread类已经实现了Runnable接口，因此，任何一个线程均有它的run方法，而run方法中包含了线程所要运行的代码。线程的活动由一组方法来控制。Java语言支持多个线程的同时执行，并提供多线程之间的同步机制（关键字为synchronized）。

- 安全性
    - Java通常被用在网络环境中，为此，Java提供了一个安全机制以防恶意代码的攻击。除了Java语言具有的许多安全特性以外，Java对通过网络下载的类具有一个安全防范机制（类ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类SecurityManager）让Java应用设置安全哨兵。

Java 历史

- 1995年5月23日，Java语言诞生
- 1996年1月，第一个JDK-JDK1.0诞生
- 1996年4月，10个最主要的操作系统供应商申明将在其产品中嵌入JAVA技术
- 1996年9月，约8.3万个网页应用了JAVA技术来制作
- 1997年2月18日，JDK1.1发布
- 1997年4月2日，JavaOne会议召开，参与者逾一万人，创当时全球同类会议规模之纪录
- 1997年9月，Java Developer Connection社区成员超过十万
- 1998年2月，JDK1.1被下载超过2,000,000次
- 1998年12月8日，Java2企业平台J2EE发布
- 1999年6月，SUN公司发布Java的三个版本：标准版（JavaSE,以前是J2SE）、企业版（JavaEE以前是J2EE）和微型版（JavaME，以前是J2ME）
- 2000年5月8日，JDK1.3发布
- 2000年5月29日，JDK1.4发布
- 2001年6月5日，NOKIA宣布，到2003年将出售1亿部支持Java的手机
- 2001年9月24日，J2EE1.3发布
- 2002年2月26日，J2SE1.4发布，自此Java的计算能力有了大幅提升
- 2004年9月30日18:00PM，J2SE1.5发布，成为Java语言发展史上的又一里程碑。为了表示该版本的重要性，J2SE1.5更名为Java SE 5.0
- 2005年6月，JavaOne大会召开，SUN公司公开Java SE 6。此时，Java的各种版本已经更名，以取消其中的数字"2"：J2EE更名为Java EE，J2SE更名为Java SE，J2ME更名为Java ME
- 2006年12月，SUN公司发布JRE6.0
- 2009年04月20日，甲骨文74亿美元收购Sun。取得Java的版权。
- 2010年11月，由于甲骨文对于Java社区的不友善，因此Apache扬言将退出JCP。
- 2011年7月28日，甲骨文发布Java7.0的正式版。

Java 语言特性

- Web开发
    - 开发大中型的网站应用。如购物商城、大型企业用系统等
- App开发
    - Android App的开发
- 游戏开发

- 数据库转换、移植
- 桌面应用开发
- 自动化测试开发

Java 环境安装

- 下载和安装JDK
- 设置环境变量
- 安装IDE开发环境

> IDE，Integrated Development Environment，集成开发环境。•一个好的编辑器或者好的IDE将会极大的提高生产力，帮我们做很多事情，使得编码工作更加简单，编码的体验更加容易。

Java 编程基础

Java 基础语法

Java是面向对象的编程语言（`尽管Java 8支持函数式编程，这篇讲义我们不考虑函数式编程`），我们学习Java，从了解`对象`开始。一个Java程序可以认为是一系列`对象`的集合，而这些对象通过调用彼此的方法来协同工作。

下面简要介绍下类、对象、方法和语句的概念。

- 对象：对象是类的一个实例，有状态和行为。例如，一只熊猫是一个对象，它的状态有：名字、年龄；行为有：卖萌、吃竹子等。
- 类：类是一个模板，一张图纸，它描述一类对象的行为和状态，把这个模板或图纸实现以后，就是一个对象。
- 方法：方法就是行为，一个类可以有很多方法。逻辑运算、数据修改以及所有动作都是在方法中完成的。卖萌、吃竹子等都是一个方法。
- 语句：语句就是每一句执行的代码。

第一个Java程序

```
public class HelloWorld {
    /* 第一个Java程序
     * 它将打印字符串 Hello World
     */
    public static void main(String []args) {
        System.out.println("Hello World Java !"); // 打印 Hello World Java !
    }
}
```

程序中包括Java的一些基本特征：

- 类(class)：上面程序定义了一个类HelloWorld，该类的名字与.java文件的名字相同。
- 方法(method)：类的内部定义了该类的一个方法main。

- 语句(statement)：真正的“打印”功能由一个语句实现，即: System.out.println("Hello World Java !");

基本语法规则

编写Java程序时，应注意以下几点：

- 大小写敏感：Java是大小写敏感的，这就意味着标识符Hello与hello是不同的。
- 类名：对于所有的类来说，类名的首字母应该大写。如果类名由若干单词组成，那么每个单词的首字母应该大写，例如 MyFirstJavaClass 。
- 方法名：所有的方法名都应该以小写字母开头。如果方法名含有若干单词，则后面的每个单词首字母大写。
- 源文件名：源文件名必须和类名相同。当保存文件的时候，你应该使用类名作为文件名保存（切记Java是大小写敏感的），文件名的后缀为.java。（如果文件名和类名不相同则会导致编译错误）。
- 主方法入口：所有的Java 程序由`public static void main(String []args)`方法开始执行。

Java 标识符

Java所有的组成部分都需要名字。类名、变量名以及方法名都被称为标识符。

关于Java标识符，有以下几点需要注意：

- 所有的标识符都应该以字母（A-Z或者a-z）,美元符（$）、或者下划线（_）开始
- 首字符之后可以是任何字符的组合
- 关键字不能用作标识符
- 标识符是大小写敏感的
- 合法标识符举例：`Panda`、`age`、`$salary`、`_value`、`__1_value`
- 非法标识符举例：`123abc`、`-salary`、`&dddd`

Java注释

类似于C/C++，Java也支持单行以及多行注释。注释中的字符将被Java编译器忽略。

```
public class HelloWorld {
    /* 这是第一个Java程序
     * 它将打印Hello World
     * 这是一个多行注释的示例
     */
    // 这是注释
    // 这里还是注释
    public static void main(String []args){
       // 这是单行注释的示例
       /* 这个也是单行注释的示例 */
       System.out.println("Hello World Java! ");
    }
}
```

Java 空行

空白行，或者有注释的的行，Java编译器都会忽略掉。

Java的对象和类

- 对象：对象是类的一个实例，有`状态`和`行为`。例如，一只熊猫是一个对象，它的状态有：名字、年龄、性别；行为有：卖萌、吃竹子等。
  - `状态`：成员变量
  - `行为`：方法
- 类：类是一个模板，一张图纸，它描述一类对象的行为和状态，把这个模板或图纸实现以后，就是一个对象。

  示例

```
class Panda{

    //成员变量 field
    String chengYuanBianLiang1 ;
    String chengYuanBianLiang2 ;
    String chengYuanBianLiang3 ;
    String chengYuanBianLiang4 ;
    String chengYuanBianLiang5 ;

    //构造方法
    Panda(){}
    Panda(String n){}
    Panda(String n, String m){}

    // 各种方法
    public void fangFa1(){
        //do someting
    }

    public void fangFa2(String n){
        //do someting
    }

    public void fangFa3(String n, String m){
        //do someting
    }

    public void fangFa4(String n, String m, String o){
        //do someting
    }
}
```

Java中的对象

现在让我们深入了解什么是对象。看看周围真实的世界，会发现身边有很多对象，车，狗，人等等。所有这些对象都有自己的状态和行为。

拿一只熊猫来举例，它的状态有：名字、性别、年龄，行为有：卖萌、改名和吃竹子。

对比现实对象和软件对象，它们之间十分相似。

软件对象也有状态和行为。软件对象的状态就是属性，行为通过方法体现。

在软件开发中，方法操作对象内部状态的改变，对象的相互调用也是通过方法来完成。

Java中的类

类可以看成是创建Java对象的模板。一张设计图纸，图纸的名字就是类的名字。

通过下面一个简单的类来理解下Java中类的定义：

```java
public class Panda {

    // 成员变量
    String pandaName = "我是个没名字的熊猫";

    // 第一种构造函数，Panda panda1 = new Panda()
    public Panda() {
        pandaName = "莹莹";
    }

    // 第二种构造函数，Panda panda2 = new Panda("贝贝")
    public Panda(String name) {
        pandaName = name;
    }


    // 方法1 卖萌
    public void maiMeng() {
        String s = "我是熊猫，我的名字是：";
        String s1 = "。我来卖个萌=￣ω￣=。";
        System.out.println(s + pandaName + s1);
    }

    // 方法2 取名，需要传递 newName 进来
    public void quMing(String newName) {
        pandaName = newName;
        System.out.println("我有名字了，我的名字是：" + pandaName + "。");
    }

    // 方法3 吃竹子，需要传递 zhuzi 进来
    public void chiZhuzi(String zhuzi) {
        // 如果是zhuzi 传递来是 好竹子，就打印
        // 定义局部变量
        String haoZhuzi = "好竹子";
        if (zhuzi == haoZhuzi) {
            String s = "我吃到了";
            System.out.println(s + haoZhuzi);
        } else {
            String s = "你不是好人啊，你竟然给我吃";
            System.out.println(s + zhuzi + "！");
        }
    }
}
```

一个类可以包含以下类型变量：

- 局部变量：在方法、构造方法或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。
- 成员变量：成员变量是定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变

量可以被类中方法、构造方法和特定类的语句块访问。

一个类可以拥有多个方法，在上面的例子中：maiMeng()、quMing()和chiZhuzi()都是Dog类的方法。

构造方法

每个类都有构造方法。如果没有显式地为类定义构造方法，Java编译器将会为该类提供一个默认构造方法。

在创建一个对象的时候，至少要调用一个构造方法。构造方法的名称必须与类同名，一个类可以有多个构造方法。

下面是一个构造方法示例：

```
public class Panda{
   public Panda(){
   }

   public Panda(String name){
      // 这个构造器仅有一个参数：name
   }
}
```

创建对象

对象是根据类创建的。在Java中，使用关键字`new`来创建一个新的对象。创建对象需要以下三步：

- 声明：声明一个对象，包括对象名称和对象类型。
- 实例化：使用关键字`new`来创建一个对象。就是根据图纸创造一个图纸描述的熊猫
- 初始化：使用`new`创建对象时，会调用构造方法初始化对象。

下面是一个创建对象的例子：

```
public class Panda{
   public Panda(String name){
      //这个构造器仅有一个参数：name
      System.out.println("Passed Name is :" + name );
   }
   public static void main(String []args){
      // 下面的语句将创建一个Panda对象
      Panda myPanda = new Panda( "tommy" );
   }
}
```

访问实例变量和方法

通过已创建的对象来访问成员变量和成员方法，如下所示：

```
/* 声明对象 + 实例化对象 */
Constructor o = new Constructor();
/* 访问其中的变量 */
o.variableName;
/* 访问类中的方法 */
o.MethodName();
```

示例

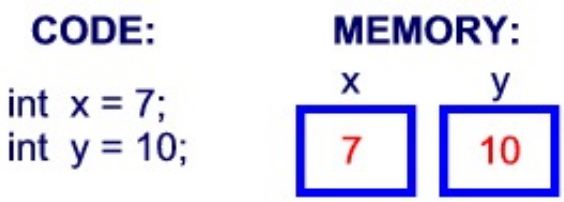
```
public class Main{

    public static void main(String []args){
    /* 创建对象 */
    // 实例化第一个熊猫，用默认的名字，莹莹
    Panda panda1 = new Panda();
    // 实例化第二个熊猫，用指定的名字：贝贝
    Panda panda2 = new Panda("贝贝");
    /* 调用一个方法获取maiMeng */
    panda1.maiMeng();
    panda2.maiMeng();
    /* 调用一个方法获取quMing */
    panda1.quMing("欢欢");
    panda1.maiMeng();
    panda1.chiZhuzi("好竹子");
    panda1.chiZhuzi("叉烧");
    panda1.chiZhuzi("红烧肉");

   }
}
```

Java 基本数据类型

变量就是申请内存来存储值。也就是说，当创建变量的时候，需要在内存中申请空间。变量存在内存中。内存管理系统根据变量的类型为变量分配存储空间，分配的空间只能用来储存该类型数据。



因此，通过定义不同类型的变量，可以在内存中储存整数、小数或者字符。

Java的两大数据类型:

- 内置数据类型
- 引用数据类型

内置数据类型

Java语言提供了八种基本类型。六种数字类型（四个整数型，两个浮点型），一种字符类型，还有一种布尔型。

byte：

- byte数据类型是8位、有符号的，以二进制补码表示的整数；
- 最小值是-128（-2^7）；
- 最大值是127（2^7-1）；
- 默认值是0；（作为类的成员变量的时候）
- byte类型用在大型数组中节约空间，主要代替整数，因为byte变量占用的空间只有int类型的四分之一；
- 例子：byte a = 100，byte b = -50。

short：

- short数据类型是16位、有符号的以二进制补码表示的整数
- 最小值是-32768（-2^15）；
- 最大值是32767（2^15 - 1）；
- Short数据类型也可以像byte那样节省空间。一个short变量是int型变量所占空间的二分之一；
- 默认值是0；
- 例子：short s = 1000，short r = -20000。

int：

- int数据类型是32位、有符号的以二进制补码表示的整数；
- 最小值是-2,147,483,648（-2^31）；
- 最大值是2,147,485,647（2^31 - 1）；
- 一般地整型变量默认为int类型；
- 默认值是0；
- 例子：int a = 100000, int b = -200000。

long：

- long数据类型是64位、有符号的以二进制补码表示的整数；
- 最小值是-9,223,372,036,854,775,808（-2^63）；
- 最大值是9,223,372,036,854,775,807（2^63 -1）；
- 这种类型主要使用在需要比较大整数的系统上；
- 默认值是0L；
- 例子： long a = 100000L，Long b = -200000L。

float：

- float数据类型是单精度、32位、符合IEEE 754标准的浮点数；
- float在储存大型浮点数组的时候可节省内存空间；
- 默认值是0.0f；
- 浮点数不能用来表示精确的值，如货币；
- 例子：float f1 = 234.5f。

double：

- double数据类型是双精度、64位、符合IEEE 754标准的浮点数；
- 浮点数的默认类型为double类型；
- double类型同样不能表示精确的值，如货币；
- 默认值是0.0d；
- 例子：double d1 = 123.4。

boolean：

- boolean数据类型表示一位的信息；
- 只有两个取值：`true`和`false`；
- 这种类型只作为一种标志来记录true/false情况；
- 默认值是false；
- 例子：boolean one = true。

char：

- char类型是一个单一的16位Unicode字符；
- 最小值是'\u0000'（即为0）；
- 最大值是'\uffff'（即为65,535）；
- char数据类型可以储存任何字符；
- 例子：char letter = 'A'。

```
public class PrimitiveTypeTest {
    public static void main(String[] args) {
        // byte
        System.out.println("基本类型：byte 二进制位数：" + Byte.SIZE);
        System.out.println("包装类：java.lang.Byte");
        System.out.println("最小值：Byte.MIN_VALUE=" + Byte.MIN_VALUE);
        System.out.println("最大值：Byte.MAX_VALUE=" + Byte.MAX_VALUE);
        System.out.println();

        // short
        System.out.println("基本类型：short 二进制位数：" + Short.SIZE);
        System.out.println("包装类：java.lang.Short");
        System.out.println("最小值：Short.MIN_VALUE=" + Short.MIN_VALUE);
        System.out.println("最大值：Short.MAX_VALUE=" + Short.MAX_VALUE);
        System.out.println();

        // int
        System.out.println("基本类型：int 二进制位数：" + Integer.SIZE);
        System.out.println("包装类：java.lang.Integer");
        System.out.println("最小值：Integer.MIN_VALUE=" + Integer.MIN_VALUE);
        System.out.println("最大值：Integer.MAX_VALUE=" + Integer.MAX_VALUE);
        System.out.println();

        // long
        System.out.println("基本类型：long 二进制位数：" + Long.SIZE);
        System.out.println("包装类：java.lang.Long");
        System.out.println("最小值：Long.MIN_VALUE=" + Long.MIN_VALUE);
        System.out.println("最大值：Long.MAX_VALUE=" + Long.MAX_VALUE);
        System.out.println();

        // float
        System.out.println("基本类型：float 二进制位数：" + Float.SIZE);
        System.out.println("包装类：java.lang.Float");
        System.out.println("最小值：Float.MIN_VALUE=" + Float.MIN_VALUE);
        System.out.println("最大值：Float.MAX_VALUE=" + Float.MAX_VALUE);
        System.out.println();

        // double
        System.out.println("基本类型：double 二进制位数：" + Double.SIZE);
        System.out.println("包装类：java.lang.Double");
        System.out.println("最小值：Double.MIN_VALUE=" + Double.MIN_VALUE);
        System.out.println("最大值：Double.MAX_VALUE=" + Double.MAX_VALUE);
        System.out.println();

        // char
        System.out.println("基本类型：char 二进制位数：" + Character.SIZE);
        System.out.println("包装类：java.lang.Character");
        // 以数值形式而不是字符形式将Character.MIN_VALUE输出到控制台
        System.out.println("最小值：Character.MIN_VALUE="
                + (int) Character.MIN_VALUE);
```

```
        // 以数值形式而不是字符形式将Character.MAX_VALUE输出到控制台
        System.out.println("最大值：Character.MAX_VALUE="
                + (int) Character.MAX_VALUE);
    }
}
```

引用类型

引用数据类型就那三种，类似C/C++的指针，它以特殊的方式指向对象实体（具体的值），这类变量声明时不会分配内存，只是存储了一个内存地址
引用数据类型包括 类，接口，数组等。

- 引用类型变量由类的构造函数创建，可以使用它们访问所引用的对象。这些变量在声明时被指定为一个特定的类型，比如Employee、Panda等。变量一旦声明后，类型就不能被改变了。
- 对象、数组都是引用数据类型。
- 所有引用类型的默认值都是`null`。
- 一个引用变量可以用来引用与任何与之兼容的类型。
- 例子：Panda myPanda = new Panda("Runoob")。

Java常量

常量在程序运行时，不会被修改的量。

在 Java 中使用 final 关键字来修饰常量，声明方式和变量类似：

```
final double PI = 3.1415927d;
```

Java 变量类型

在Java语言中，所有的变量在使用前必须声明。声明变量的基本格式如下：

```
int a, b, c;         // 声明三个int型整数：a、 b、c
int d = 3, e, f = 5; // 声明三个整数并赋予初值
byte z = 22;         // 声明并初始化 z
String s = "runoob"  // 声明并初始化字符串 s
double pi = 3.14159d; // 声明了双精度浮点型变量 pi
char x = 'x';        // 声明变量 x 的值是字符 'x'。

int i=97;
System.out.println((char)i);
```

Java 局部变量

- 局部变量声明在方法、构造方法或者语句块中；
- 局部变量在方法、构造方法、或者语句块被执行的时候创建，当它们执行完成后，变量将会被销毁；
- 访问修饰符不能用于局部变量；

- 局部变量只在声明它的方法、构造方法或者语句块中可见；
- 局部变量是在栈上分配的。
- 局部变量没有默认值，所以局部变量被声明后，必须经过初始化，才可以使用

```
import package com.runoob.test;

public class Test{
   public void pupAge(){
      int age = 0;
      age = age + 7;
      System.out.println("小狗的年龄是: " + age);
   }

   public static void main(String args[]){
      Test test = new Test();
      test.pupAge();
   }
}



2
  public class calc{
    int sum =0;
    public int add1(int a,int b){
         sum =a+b;
         return sum;
    }
    public int add2(int a,int b){
         sum =sum+a+b;
         return sum;
    }
    public static void main(String[] args)
    {
         calc ca =new calc();
         int i = ca.add1(12,13);
         i =ca.add2(100,200);
      System.out.println(i);
    }
}
```

实例变量

- 实例变量声明在一个类中，但在方法、构造方法和语句块之外；
- 当一个对象被实例化之后，每个实例变量的值就跟着确定；
- 实例变量在对象创建的时候创建，在对象被销毁的时候销毁；
- 实例变量的值应该至少被一个方法、构造方法或者语句块引用，使得外部能够通过这些方式获取实例变量信息；
- 实例变量可以声明在使用前或者使用后；

- 访问修饰符可以修饰实例变量；
- 实例变量对于类中的方法、构造方法或者语句块是可见的。一般情况下应该把实例变量设为私有。通过使用访问修饰符可以使实例变量对子类可见；
- 实例变量具有默认值。数值型变量的默认值是0，布尔型变量的默认值是false，引用类型变量的默认值是null。变量的值可以在声明时指定，也可以在构造方法中指定；
- 实例变量可以直接通过变量名访问。但在静态方法以及其他类中，就应该使用完全限定名：ObejectReference.VariableName。

实例：

```
import java.io.*;
public class Employee{
   // 这个成员变量对子类可见
   public String name;
   // 私有变量，仅在该类可见
   private double salary;
   //在构造器中对name赋值
   public Employee (String empName){
      name = empName;
   }
   //设定salary的值
   public void setSalary(double empSal){
      salary = empSal;
   }
   // 打印信息
   public void printEmp(){
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]){
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

Java语言提供了很多修饰符，主要分为以下两类：

- 访问修饰符
- 非访问修饰符

修饰符用来定义类、方法或者变量，通常放在语句的最前端。我们通过下面的例子来说明：

Java String类

字符串广泛应用在Java编程中，在Java中字符串属于对象，Java提供了String类来创建和操作字符串。

```
String hello = "Hello world!";
```

字符串长度

用于获取有关对象的信息的方法称为访问器方法。

String类的一个访问器方法是length()方法，它返回字符串对象包含的字符数。

下面的代码执行后，len变量等于17:

```
public static void main(String args[]) {
  String palindrome = "Dot saw I was Tod";
  int len = palindrome.length();
  System.out.println( "String Length is : " + len );
}
```

连接字符串

String类提供了连接两个字符串的方法：

```
string1.concat(string2);
string1 + string2
```

示例

```
"Hello ".concat("World!");
"Hello " + "World!"


        String str1 ="hello",str2="world";
        int len=str1.length();
        System.out.println(len);
        str1=str1.concat(str2);
       // str1=str1.concat(str2).concat(str1);
        System.out.println(str1);
```

s

创建格式化字符串

我们知道输出格式化数字可以使用printf()和format()方法。String类使用静态方法format()返回一个String对象而不是PrintStream对象。

String类的静态方法format()能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出。如下所示：

```
String fs;
fs = String.format("The value of the float variable is " +
                   "%f, while the value of the integer " +
                   "variable is %d, and the string " +
                   "is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

```
        String name ="zhangsan";
        int age =18;
        String str =String.format("you name is %s,you age is %d", name,age);
        System.out.println(str);
```

String字符串的方法

方法名称	描述
`s.length()`	返回s字符串长度
`s.charAt(2)`	返回s字符串中下标为2的字符
`s.substring(0, 4)`	返回s字符串中下标0到4的子字符串
`s.indexOf("Hello")`	返回子字符串"Hello"的下标
`s.startsWith(" ")`	判断s是否以空格开始
`s.endsWith("oo")`	判断s是否以"oo"结束
`s.equals("Good World!")`	判断s是否等于"Good World!"
`s.trim()`	去掉s前后的空格字符串，并返回新的字符串
`s.toUpperCase()`	将s转换为大写字母，并返回新的字符串
`s.toLowerCase()`	将s转换为小写，并返回新的字符串

```
    String name ="yaoming";
    char c = name.charAt(2);
    System.out.println(c);
    String str =name.substring(0,4);
    System.out.println(str);
    System.out.println("---返回字符串的下标");
    System.out.println(name.indexOf("mi"));
    System.out.println("判断s是否以空格开始");
    System.out.println(name.startsWith(" "));
    System.out.println("判断s是否以ing结束");
    System.out.println(name.endsWith("ing"));
    System.out.println("判断s是否等于");
    System.out.println(name.equals("yaoming"));

    String myname ="  TangXingLong  ";
    //掉s前后的空格字符串，并返回新的字符串
    System.out.println(myname.trim());
    //将s转换为大写字母，并返回新的字符串
    System.out.println(myname.toUpperCase());
    //将s转换为小写，并返回新的字符串
    System.out.println(myname.toLowerCase());
```

Java 数组

数组对于每一门编程语言来说都是重要的数据结构之一，当然不同语言对数组的实现及处理也不尽相同。

Java语言中提供的数组是用来存储固定大小的同类型元素。

你可以声明一个数组变量，如numbers[100]来代替直接声明100个独立变量number0，number1，....，number99。

本教程将为大家介绍Java数组的声明、创建和初始化，并给出其对应的代码。

声明数组变量

首先必须声明数组变量，才能在程序中使用数组。下面是声明数组变量的语法：

```
DataType[] array;   // 首选的方法
```

实例

下面是这两种语法的代码示例：

```
double[] myList;
int[] myList1;
float[] myList2;
char[] myList3;
String[] myList4;
```

```
    #先声明，在赋值
    int[] myList;
    myList = new int[5];
    myList[0]=10;
    myList[1]=1;
    System.out.println(myList[0]+myList[1]);
```

```
    int[] myList=new int[5];
    myList[0]=100;
    myList[1]=101;
    System.out.print(myList[0]+myList[1]);
```

```
        int[] myList={1,2,30,4,5};
        //打印数组的值
//      for(int i=0;i<=4;i++)
//          System.out.println(myList[i]+"  ");

        //数组所有元素之和
        int total=0;
        for(int i=0;i<myList.length;i++)
            total =total + myList[i];
        System.out.println("total="+total);
        //查找最大元素
        int max =myList[0];
        for(int i=0;i<myList.length;i++)
            if(myList[i]>max)
                max = myList[i];
        System.out.println("max="+max);
```

```
       //数组互相赋值
       int[] myList={1,2,30,4,5};
   int[] myList1;
   myList1=myList;
   System.out.println(myList[0]);
```

创建数组

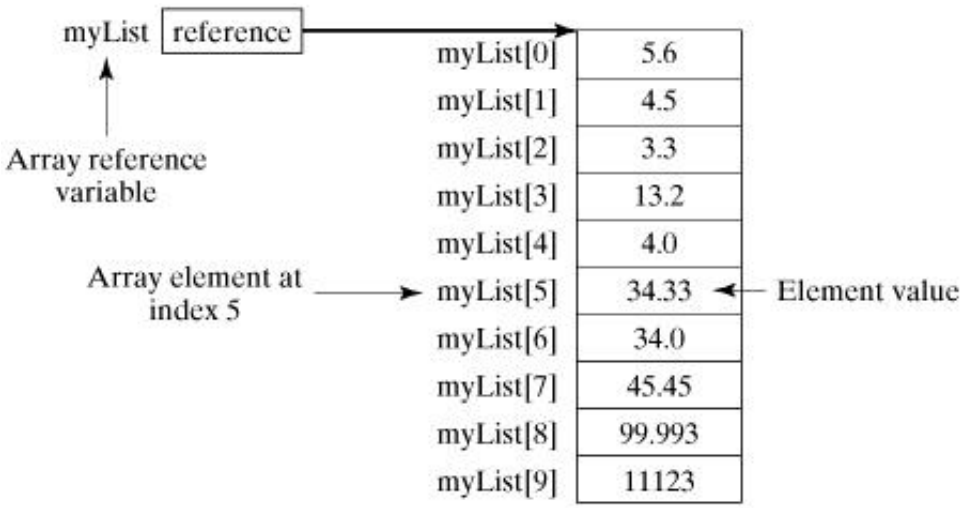
Java语言使用new操作符来创建数组，语法如下：

```
myList = new double[10]
```

示例

```java
public static void main(String[] args) {
      // 数组大小
      //int size = 10;
      // 定义数组
      double[] myList = new double[size];
      myList[0] = 5.6;
      myList[1] = 4.5;
      myList[2] = 3.3;
      myList[4] = 4.0;
      myList[5] = 34.33;
      myList[6] = 34.0;
      myList[7] = 45.45;
      myList[8] = 99.993;
      myList[9] = 11123;
      // 计算所有元素的总和
      double total = 0;
      for (int i = 0; i < size; i++) {
         total += myList[i];
      }
      System.out.println("总和为：  " + total);
   }
```

下面的图片描绘了数组myList。这里myList数组里有10个double元素，它的下标从0到9。



处理数组

数组的元素类型和数组的大小都是确定的，所以当处理数组元素时候，我们通常使用基本循环

```
public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // 打印所有数组元素
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }
      // 计算所有元素的总和
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // 查找最大元素
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) {
           max = myList[i];
         }
      }
      System.out.println("Max is " + max);
   }
```

数组作为函数的参数

函数前加static与不加的区别：

> 就是调用时候方便 不加static是非静态函数，访问需要new出该类的对象来调用，加上static是静态函数 可直接访问或者通过类名访问

数组可以作为参数传递给方法。例如，下面的例子就是一个打印 `int` 数组中元素的方法。

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

```
public static void printArray(int[] array) {
       for (int i = 0; i < array.length; i++) {
         System.out.print(array[i] + " ");
       }
     }
public static void main(String[] args)
{
     int[] mylist={10,20,30,40,50}    ;
     printArray(mylist);
}
```

数组作为函数的返回值

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```

java string 数组赋值

```
        String[] sale1 = {"111", "5555"};
        String[] sale;
        sale = new String[10];
        sale[0]="aaaa";
        System.out.println(sale.length);
        System.out.println(sale[0].length());
        System.out.println(sale1.length);
        System.out.println(sale1[0].length());
```

Java 算术运算符

算术运算符用在数学表达式中，它们的作用和在数学中的作用一样。下表列出了所有的算术运算符。

表格中的实例假设整数变量A的值为10，变量B的值为20：

操作符	描述	例子
+	加法 - 相加运算符两侧的值	A + B等于30
-	减法 - 左操作数减去右操作数	A – B等于-10
*	乘法 - 相乘操作符两侧的值	A * B等于200
/	除法 - 左操作数除以右操作数	B / A等于2
%	取模 - 右操作数除左操作数的余数	B % A等于0
++	自增 - 操作数的值增加1	B + +等于21
--	自减 -- 操作数的值减少1	B - -等于19

实例

下面的简单示例程序演示了算术运算符。复制并粘贴下面的Java程序并保存为Test.java文件，然后编译并运行这个程序：

```
public class Test {

  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     int c = 25;
     int d = 25;
     System.out.println("a + b = " + (a + b) );
     System.out.println("a - b = " + (a - b) );
     System.out.println("a * b = " + (a * b) );
     System.out.println("b / a = " + (b / a) );
     System.out.println("b % a = " + (b % a) );
     System.out.println("c % a = " + (c % a) );
     System.out.println("a++   = " +  (a++) );
     System.out.println("a--   = " +  (a--) );
     // 查看  d++ 与 ++d 的不同
     System.out.println("d++   = " +  (d++) );
     System.out.println("++d   = " +  (++d) );
  }
}
```

Java 循环结构 - for, while 及 do...while

顺序结构的程序语句只能被执行一次。如果您想要同样的操作执行多次,，就需要使用循环结构。

Java中有三种主要的循环结构：

- while循环
- do...while循环

- for循环

while循环

while是最基本的循环，它的结构为：

```
while( 布尔表达式 ) {
    //循环内容
}
```

只要布尔表达式为true，循环体会一直执行下去。

实例

```
public class Test {
   public static void main(String args[]) {
      int x = 10;
      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

do...while循环

对于while语句而言，如果不满足条件，则不能进入循环。但有时候我们需要即使不满足条件，也至少执行一次。

do...while循环和while循环相似，不同的是，do...while循环至少会执行一次。

```
do {
       //代码语句
}while(布尔表达式);
```

注意：布尔表达式在循环体的后面，所以语句块在检测布尔表达式之前已经执行了。 如果布尔表达式的值为true，则语句块一直执行，直到布尔表达式的值为false。

实例

```
public class Test {

   public static void main(String args[]){
      int x = 10;

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

for循环

虽然所有循环结构都可以用while或者do...while表示，但Java提供了另一种语句 —— for循环，使一些循环结构变得更加简单。

for循环执行的次数是在执行前就确定的。语法格式如下：

```
for(初始化; 布尔表达式; 更新) {
    //代码语句
}
```

关于for循环有以下几点说明：

- 最先执行初始化步骤。可以声明一种类型，但可初始化一个或多个循环控制变量，也可以是空语句。
- 然后，检测布尔表达式的值。如果为true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。
- 执行一次循环后，更新循环控制变量。
- 再次检测布尔表达式。循环执行上面的过程。

实例

```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

九九乘法表

```
for(int i=1;i<=9;i++){
            for(int j=1;j<=i;j++){
                System.out.print(j + "*" + i +"="+i*j+" ");
            }
            System.out.println("");
        }
```

break关键字

break主要用在循环语句或者switch语句中，用来跳出整个语句块。

break跳出最里层的循环，并且继续执行该循环下面的语句。

语法

break的用法很简单，就是循环结构中的一条语句：

```
break;
```

实例

```
       for(int i=1;i<=10;i++){
           if(i==5)
               break;
           System.out.println(i);
       }


    public static void main(String[] args) {
       for(int i=1;i<=10;i++){
           for(int j=1;j<=10;j++){
               if(j==5)
                   break;
               System.out.print("* ");
           }
           System.out.print("\n");
       }
    }
```

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

continue关键字

continue适用于任何循环控制结构中。作用是让程序立刻跳转到下一次循环的迭代。

在for循环中，continue语句使程序立即跳转到更新语句。

在while或者do...while循环中，程序立即跳转到布尔表达式的判断语句。

语法

continue就是循环体中一条简单的语句：

```
continue;
```

实例

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

Java 分支结构 - if...else/switch

顺序结构只能顺序执行，不能进行判断和选择，因此需要分支结构。

Java有两种分支结构：

- if语句
- switch语句

if语句

一个if语句包含一个布尔表达式和一条或多条语句。

语法

if语句的用语法如下：

```
if(布尔表达式)
{
   //如果布尔表达式为true将执行的语句
}
```

如果布尔表达式的值为true，则执行if语句中的代码块。否则执行If语句块后面的代码。

```
public class Test {

   public static void main(String args[]){
      int x = 10;

      if( x < 20 ){
         System.out.print("这是 if 语句");
      }
   }
}
```

if...else语句

if语句后面可以跟else语句，当if语句的布尔表达式值为false时，else语句块会被执行。

语法

if...else的用法如下：

```
if(布尔表达式){
   //如果布尔表达式的值为true
}else{
   //如果布尔表达式的值为false
}
```

实例

```
public class Test {

   public static void main(String args[]){
      int x = 30;

      if( x < 20 ){
         System.out.print("这是 if 语句");
      }else{
         System.out.print("这是 else 语句");
      }
   }
}
```

switch语句

```
switch语句判断一个变量与一系列值中某个值是否相等，每个值称为一个分支。

语法

switch语法格式如下：

switch(expression){
    case value :
       //语句
       break; //可选
    case value :
       //语句
       break; //可选
    //你可以有任意数量的case语句
    default : //可选
       //语句
}
```

switch语句有如下规则：

- switch语句中的变量类型只能为byte、short、int或者char。
- switch语句可以拥有多个case语句。每个case后面跟一个要比较的值和冒号。
- case语句中的值的数据类型必须与变量的数据类型相同，而且只能是常量或者字面常量。
- 当变量的值与case语句的值相等时，那么case语句之后的语句开始执行，直到break语句出现才会跳出switch语句。
- 当遇到break语句时，switch语句终止。程序跳转到switch语句后面的语句执行。case语句不必须要包含break语句。如果没有break语句出现，程序会继续执行下一条case语句，直到出现break语句。
- switch语句可以包含一个default分支，该分支必须是switch语句的最后一个分支。default在没有case语句的值和变量值相等的时候执行。default分支不需要break语句。

实例

```
public class Test {

   public static void main(String args[]){
      //char grade = args[0].charAt(0);
      char grade = 'C';

      switch(grade)
      {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

java的输入
首先，导入java.util.*包。

```
import java.util.Scanner;
```

新建一个读取标准输入（键盘）的扫描器对象。

```
Scanner in = new Scanner(System.in);
```

获取输入的字符串

```
String s = in.nextLine();
```

获取输入的整数

```
int i_var = a.nextInt();
```

获取输入double类型值

```
double f = a.nextDouble();
```

输入两个数，然后选择运算符（+，-，*，/），实现小计算器的功能

```
      public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
        System.out.println("输入第一个数");
        double d1 = input.nextDouble();
        System.out.println("输入第二个值");
        double d2 =input.nextDouble();
        System.out.println("请选择运算符+，-，*，/");
        input.nextLine();
        String s = input.nextLine();
        //获取字符串中的第一个字符
        char c = s.charAt(0);
        switch(c){
            case '+':
                System.out.println("两数之和为："+(d1+d2));
                break;
            case '-':
                System.out.println("两数之差为："+(d1-d2));
                break;
            case '*':
                System.out.println("两数之积为："+(d1 * d2));
                break;
            case '/':
                System.out.println("两数之商为："+(d1/d2));
                break;
            default:
                System.out.println("输入的运算符错误");
        }

    }
```

Java 面向对象

Java 方法

在前面几个章节中我们经常使用到System.out.println()，那么它是什么呢？

println()是一个方法(Method)，而System是系统类(Class)，out是标准输出对象(Object)。这句话的用法是调用系统类System中的标准输出对象out中的方法println()。

那么什么是方法呢？

Java方法是语句的集合，它们在一起执行一个功能。

- 方法是解决一类问题的步骤的有序组合
- 方法包含于类或对象中
- 方法在程序中被创建，在其他地方被引用

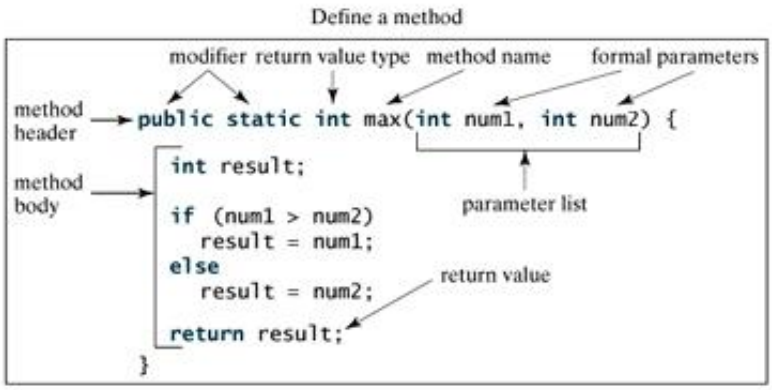
方法的定义

一般情况下，定义一个方法包含以下语法：

```
修饰符 返回值类型 方法名 (参数类型 参数名){
    ...
    方法体
    ...
    return 返回值;
}
```

方法包含一个方法头和一个方法体。下面是一个方法的所有部分：

- 修饰符：修饰符，这是可选的，告诉编译器如何调用该方法。定义了该方法的访问类型。
- 返回值类型 ：方法可能会返回值。returnValueType是方法返回值的数据类型。有些方法执行所需的操作，但没有返回值。在这种情况下，returnValueType是关键字**void**。
- 方法名：是方法的实际名称。方法名和参数表共同构成方法签名。
- 参数类型：参数像是一个占位符。当方法被调用时，传递值给参数。这个值被称为实参或变量。参数列表是指方法的参数类型、顺序和参数的个数。参数是可选的，方法可以不包含任何参数。
- 方法体：方法体包含具体的语句，定义该方法的功能。



如：

```
public static int age(int birthday){...}
```

参数可以有多个：

```
static float interest(float principal, int year){...}
```

注意： 在一些其它语言中方法指过程和函数。一个返回非void类型返回值的方法称为函数；一个返回void类型返回值的方法叫做过程。

实例

下面的方法包含2个参数num1和num2，它返回这两个参数的最大值。

```
/** 返回两个整型变量数据的较大值 */
public static int max(int num1, int num2) {
   int result;
   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

方法调用

Java支持两种调用方法的方式，根据方法是否返回值来选择。

当程序调用一个方法时，程序的控制权交给了被调用的方法。当被调用方法的返回语句执行或者到达方法体闭括号时候交还控制权给程序。

当方法返回一个值的时候，方法调用通常被当做一个值。例如：

```
int larger = max(30, 40);
```

如果方法返回值是void，方法调用一定是一条语句。例如，方法println返回void。下面的调用是个语句：

```
System.out.println("Welcome to Java!");
```

示例

下面的例子演示了如何定义一个方法，以及如何调用它：

```
public class TestMax {
   /** 主方法 */
   public static void main(String[] args) {
      int i = 5;
      int j = 2;
      int k = max(i, j);
      System.out.println("The maximum between " + i +
                  " and " + j + " is " + k);
   }

   /** 返回两个整数变量较大的值 */
   public static int max(int num1, int num2) {
      int result;
      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
}
```

这个程序包含main方法和max方法。Main方法是被JVM调用的，除此之外，main方法和其它方法没什么区别。

main方法的头部是不变的，如例子所示，带修饰符public和static,返回void类型值，方法名字是main,此外带个一个String[]类型参数。String[]表明参数是字符串数组。

void 关键字

本节说明如何声明和调用一个void方法。

下面的例子声明了一个名为printGrade的方法，并且调用它来打印给定的分数。

示例

```
public class TestVoidMethod {

   public static void main(String[] args) {
      printGrade(78.5);
   }

   public static void printGrade(double score) {
      if (score >= 90.0) {
         System.out.println('A');
      }
      else if (score >= 80.0) {
         System.out.println('B');
      }
      else if (score >= 70.0) {
         System.out.println('C');
      }
      else if (score >= 60.0) {
         System.out.println('D');
      }
      else {
         System.out.println('F');
      }
   }
}
```

这里printGrade方法是一个void类型方法，它不返回值。

一个void方法的调用一定是一个语句。 所以，它被在main方法第三行以语句形式调用。就像任何以分号结束的语句一样。

通过值传递参数

调用一个方法时候需要提供参数，你必须按照参数列表指定的顺序提供。

例如，下面的方法连续n次打印一个消息：

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

示例

下面的例子演示按值传递的效果。

该程序创建一个方法，该方法用于交换两个变量。

```
public class TestPassByValue {

   public static void main(String[] args) {
      int num1 = 1;
      int num2 = 2;

      System.out.println("Before swap method, num1 is " +
                         num1 + " and num2 is " + num2);

      // 调用swap方法
      swap(num1, num2);
      System.out.println("After swap method, num1 is " +
                        num1 + " and num2 is " + num2);
   }
   /** 交换两个变量的方法 */
   public static void swap(int n1, int n2) {
      System.out.println("\tInside the swap method");
      System.out.println("\t\tBefore swapping n1 is " + n1
                          + " n2 is " + n2);
      // 交换 n1 与 n2的值
      int temp = n1;
      n1 = n2;
      n2 = temp;

      System.out.println("\t\tAfter swapping n1 is " + n1
                          + " n2 is " + n2);
   }
}
```

访问控制修饰符

Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java支持4种不同的访问权限。

默认的，也称为 **default**，在同一包内可见，不使用任何修饰符。

私有的，以 **private** 修饰符指定，在同一类内可见。

共有的，以 **public** 修饰符指定，对所有类可见。

受保护的，以 **protected** 修饰符指定，对同一包内的类和所有子类可见。

默认访问修饰符-不使用任何关键字

使用默认访问修饰符声明的变量和方法，对同一个包内的类是可见的。接口里的变量都隐式声明为public static final,而接口里的方法默认情况下访问权限为public。

实例：

如下例所示，变量和方法的声明可以不使用任何修饰符。

```
String version = "1.5.1";
boolean processOrder() {
   return true;
}
```

私有访问修饰符**-private**

私有访问修饰符是最严格的访问级别，所以被声明为private的方法、变量和构造方法只能被所属类访问，并且类和接口不能声明为private。

声明为私有访问类型的变量只能通过类中公共的getter方法被外部类访问。

Private访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据。

下面的类使用了私有访问修饰符：

```
public class Logger {
   private String format;
   public String getFormat() {
      return this.format;
   }
   public void setFormat(String format) {
      this.format = format;
   }
}
```

实例中，Logger类中的format变量为私有变量，所以其他类不能直接得到和设置该变量的值。为了使其他类能够操作该变量，定义了两个public方法：getFormat() （返回format的值）和setFormat(String)（设置format的值）

公有访问修饰符**-public**

被声明为public的类、方法、构造方法和接口能够被任何其他类访问。

如果几个相互访问的public类分布在不同的包中，则需要导入相应public类所在的包。由于类的继承性，类所有的公有方法和变量都能被其子类继承。

以下函数使用了公有访问控制：

```
public static void main(String[] arguments) {
   // ...
}
```

Java程序的main() 方法必须设置成公有的，否则，Java解释器将不能运行该类。

受保护的访问修饰符**-protected**

被声明为protected的变量、方法和构造器能被同一个包中的任何其他类访问，也能够被不同包中的子类访问。

Protected访问修饰符不能修饰类和接口，方法和成员变量能够声明为protected，但是接口的成员变量和成员方法不能声明为protected。

子类能访问Protected修饰符声明的方法和变量，这样就能保护不相关的类使用这些方法和变量。

下面的父类使用了protected访问修饰符，子类重载了父类的openSpeaker()方法。

```
class AudioPlayer {
   protected boolean openSpeaker(Speaker sp) {
      // 实现细节
   }
}

class StreamingAudioPlayer {
   boolean openSpeaker(Speaker sp) {
      // 实现细节
   }
}
```

如果把openSpeaker()方法声明为private，那么除了AudioPlayer之外的类将不能访问该方法。如果把openSpeaker()声明为public，那么所有的类都能够访问该方法。如果我们只想让该方法对其所在类的子类可见，则将该方法声明为protected。

访问控制和继承

请注意以下方法继承的规则：

- 父类中声明为public的方法在子类中也必须为public。
- 父类中声明为protected的方法在子类中要么声明为protected，要么声明为public。不能声明为private。
- 父类中声明为private的方法，不能够被继承。

Java 继承

继承是java面向对象编程技术的一块基石，因为它允许创建分等级层次的类。继承可以理解为一个对象从另一个对象获取属性的过程。

如果类A是类B的父类，而类B是类C的父类，我们也称C是A的子类，类C是从类A继承而来的。在Java中，类的继承是单一继承，也就是说，一个子类只能拥有一个父类

继承中最常使用的两个关键字是`extends`和`implements`。

这两个关键字的使用决定了一个对象和另一个对象是否是IS-A(是一个)关系。

通过使用这两个关键字，我们能实现一个对象获取另一个对象的属性。

所有Java的类均是由`java.lang.Object`类继承而来的，所以`Object`是所有类的祖先类，而除了Object外，所有类必须有一个父类。

通过过extends关键字可以申明一个类是继承另外一个类而来的，一般形式如下：

```
// A.java
public class A {
    private int i;
    protected int j;

    public void func() {

    }
}

// B.java
public class B extends A {
}
```

以上的代码片段说明，B由A继承而来的，B是A的子类。而A是Object的子类，这里可以不显示地声明。

作为子类，B的实例拥有A所有的成员变量，但对于private的成员变量B却没有访问权限，这保障了A的封装性。

IS-A关系

IS-A就是说:一个对象是另一个对象的一个分类。

下面是使用关键字extends实现继承。

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

基于上面的例子，以下说法是正确的：

- Animal类是Mammal类的父类。
- Animal类是Reptile类的父类。
- Mammal类和Reptile类是Animal类的子类。
- Dog类既是Mammal类的子类又是Animal类的子类。

分析以上示例中的IS-A关系，如下：

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal

因此 : Dog IS-A Animal

通过使用关键字**extends**，子类可以继承父类所有的方法和属性，但是无法使用 private(私有) 的方法和属性。

我们通过使用instanceof 操作符，能够确定Mammal IS-A Animal

实例

```
public class Dog extends Mammal{

   public static void main(String args[]){

      Animal a = new Animal();
      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
   }
}
```

介绍完`extends`关键字之后，我们再来看下`implements`关键字是怎样使用来表示IS-A关系。

Implements关键字使用在类继承接口的情况下， 这种情况不能使用关键字**extends**。

实例

```
public interface Animal {}

public class Mammal implements Animal{
}

public class Dog extends Mammal{
}
```

比如：People是一个接口，他里面有say这个方法。
接口的定义：

```
public interface People{
public void say();
}
```

但是接口没有方法体。只能通过一个具体的类去实现其中的方法体。
比如 Chinese这个类，就实现了People这个接口。
接口的实现：

```
public class Chinese implements People{
public void say() {
System.out.println(" 你好！ ");
}
}
```

接口的调用：

```
People chinese = new Chinese() ;
chinese.say();
```

冒泡排序

> 冒泡排序（Bubble Sort）是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。

冒泡排序算法的运作如下:

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

代码如下：

```
public class BubbleSort {
    public void doSort() {
        int score[] = {67, 69, 75, 87, 89, 90, 99, 100};
        //最多做n-1趟排序
        for (int i = 0; i < score.length - 1; i++) {
            //对当前无序区间score[0......length-i-1]进行排序(j的范围很关键，这个范围是在逐步缩
小的)
            for (int j = 0; j < score.length - i - 1; j++) {
                //把小的值交换到后面
                /*if (score[j] < score[j + 1]) {
                    int temp = score[j];
                    score[j] = score[j + 1];
                    score[j + 1] = temp;
                }*/
                int tmp = pickSmaller(score[j], score[j + 1]);
                score[j] = pickGreater(score[j], score[j + 1]);
                score[j + 1] = tmp;
            }
            System.out.print("第" + (i + 1) + "次排序结果：");
            for (int a = 0; a < score.length; a++) {
                System.out.print(score[a] + "\t");
            }
            System.out.println("");
        }
        System.out.print("最终排序结果：");
        for (int a = 0; a < score.length; a++) {
            System.out.print(score[a] + "\t");
        }
    }

    public int pickGreater(int a, int b) {
        if (a > b) {
            return a;

        }
        return b;
    }

    public int pickSmaller(int a, int b) {
        if (a < b) {
            return a;
        }
        return b;
    }
}
```